Problem 1 Multi-threaded Event Ticketing System

Problem Statement

You are tasked with designing and implementing a multi-threaded event ticketing system that simulates the operations of an event management platform. The system should handle multiple users (threads) performing various operations such as purchasing tickets, viewing event details, and managing user accounts. The system must ensure data consistency and prevent issues like deadlocks and starvation.

Requirements

1. **Event Management**:
    - Add new events.
    - Update event details.
    - Remove events.
2. **User Management**:
    - Register new users.
    - Login and logout operations.
3. **Ticket Management**:
    - Purchase tickets for an event.
    - View available tickets for an event.
    - Cancel purchased tickets.
4. **Concurrency Control**:
    - Use read-write locks to manage access to event details.
    - Use reentrant locks to handle nested function calls.
    - Use conditional variables to manage ticket availability notifications.
    - Use try locks to handle concurrent ticket purchase attempts.
5. **Liveness**:
    - Ensure the system is free from deadlocks.
    - Prevent starvation by ensuring all users get a fair chance to perform operations.

Menu Options

The system should provide a menu-driven interface for users to perform the following operations:

1. **Event Management**:
    - **Add Event**: Add a new event to the system.
    - **Update Event**: Update details of an existing event.
    - **Remove Event**: Remove an event from the system.
2. **User Management**:
    - **Register User**: Register a new user.
    - **Login**: Login to the system.
    - **Logout**: Logout from the system.
3. **Ticket Management**:
    - **Purchase Ticket**: Purchase a ticket for an event.
    - **View Tickets**: View available tickets for an event.
    - **Cancel Ticket**: Cancel a purchased ticket.
4. **Concurrency Control**:
    - **Display Lock Status**: Display the status of locks (e.g., which resources are locked).
5. **Liveness**:
    - **Check Deadlocks**: Check for potential deadlocks and resolve them.
    - **Ensure Fairness**: Ensure all users get a fair chance to perform operations.

Problem 2 - Multi-threaded Smart Home System

Problem Statement

You are tasked with designing and implementing a multi-threaded smart home system that simulates the operations of a smart home environment. The system should handle multiple users (threads) performing various operations such as controlling smart devices, monitoring device status, and managing user accounts. The system must ensure data consistency and prevent issues like deadlocks and starvation.

Requirements

1. **Device Management**:
    - Add new devices.
    - Update device details.
    - Remove devices.
2. **User Management**:
    - Register new users.
    - Login and logout operations.
3. **Device Control**:
    - Turn devices on/off.
    - Adjust device settings (e.g., temperature for thermostats).
    - Monitor device status.
4. **Concurrency Control**:
    - Use read-write locks to manage access to device details.
    - Use reentrant locks to handle nested function calls.
    - Use conditional variables to manage device status notifications.
    - Use try locks to handle concurrent device control attempts.
5. **Liveness**:
    - Ensure the system is free from deadlocks.
    - Prevent starvation by ensuring all users get a fair chance to perform operations.

Menu Options

The system should provide a menu-driven interface for users to perform the following operations:

1. **Device Management**:
   - **Add Device**: Add a new device to the system.
   - **Update Device**: Update details of an existing device.
   - **Remove Device**: Remove a device from the system.
2. **User Management**:
   - **Register User**: Register a new user.
   - **Login**: Login to the system.
   - **Logout**: Logout from the system.
3. **Device Control**:
   - **Turn Device On/Off**: Control the power state of a device.
   - **Adjust Device Settings**: Adjust settings for a device.
   - **Monitor Device Status**: View the current status of a device.
4. **Concurrency Control**:
   - **Display Lock Status**: Display the status of locks (e.g., which resources are locked).
5. **Liveness**:
   - **Check Deadlocks**: Check for potential deadlocks and resolve them.
   - **Ensure Fairness**: Ensure all users get a fair chance to perform operations.